



aka MIPS Procedures

*CS31*

*Pascal Van Hentenryck*



# MIPS Procedures

Functions in C, C++

Methods in Java

- A method is just a function with the receiver object as the first arguments

Outline

- Branching and return
- Passing arguments
- Saving registers

# Procedures

## Procedure without arguments

- Where to jump?
- Where to go back?

```
__start:      move    $s0,$s1
              ...
              {call procedure gumbo}
mul          $s2,$s5,$s7
              ...
              done

gumbo:       add     $s3,$s4,$s5
              ...
              {go back to where we came from}
```

# The Kick

How do I go back now?



# Jumping and Kicking

MAL helps us quite a bit

```
jal    label
```

- puts the address of the next instruction into register \$ra (return address)
- branches to label

This is easy to do in hardware since the PC contains the right address (or almost)

# Example Procedure

```
__start:    li    $s0, 7
            jal    verse1
            jal    refrain
            jal    verse2
            jal    refrain
            done

verse1:     ...
            ...
            jr     $ra

refrain:    ...
            ...
            jr     $ra
```

Could we do without `jal`?

# Example Procedure

```
start:      li    $s0, 7
            jal    verse1
            jal    refrain
            jal    verse2
            jal    refrain
            done

verse1:     ...
            jal    subverse1
            li     $s0, 33
            jr     $ra

refrain:    ...
            ...
            jr     $ra
```

- What if `verse1` does a `jal`?
- What if it uses `$s0`?

# Example Procedure

```
start:      li    $s0, 7
            jal   verse1
            jal   refrain
            jal   verse2
            jal   refrain
            done
verse1:     ...
            jal   subverse1
            li    $s0, 33
            jr    $ra
```

- What if `verse1` does a `jal`?
- What if it uses `$s0`?



# Example Procedure

The 5 Levels Of <b>INCEPTION</b>				
LEVEL	WHO DREAMED IT?	WHO GOES THERE?	WHY ARE THEY THERE?	THE KICK
<b>LEVEL 1</b> <b>REALITY</b>	<b>No one...</b> We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
<b>LEVEL 2</b> <b>VAN CHASE</b>	<b>Yusuf</b> "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fischer Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
<b>LEVEL 3</b> <b>THE HOTEL</b>	<b>Arthur</b> "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.
<b>LEVEL 4</b> <b>SNOW FORTRESS</b>	<b>Eames</b> "The Forger"	Cobb, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. must be taken to the fort, where the idea they wish to plant will finally take hold.	Eames blows up the supports of the fortress, dropping it and causing freefall.
<b>LEVEL 5</b> <b>LIMBO</b>	<b>No one</b> It's a shared state	Cobb, Ariadne, Saito, Robert Fischer Jr. and Mal's projection	To get Fischer Jr. and Saito out.	Ariadne and Fischer fall off a building. Cobb and Saito shoot themselves.

# The System Stack

(A Necessary Digression)

Sometimes we have to save data into memory:

- return addresses for nested procedures
- register values if more than one procedure wants to use the same register

It's inconvenient to have to anticipate exactly how much such storage we'll need and allocate memory to it explicitly.

Instead, we'll construct another way to allocate memory locations: the system stack.

# Stacks in the Abstract

Stacks have two operations:

- push(item): add item to the top of the stack
- pop: remove the item from the top of the stack

push a

push b

push c

pop =>

pop =>

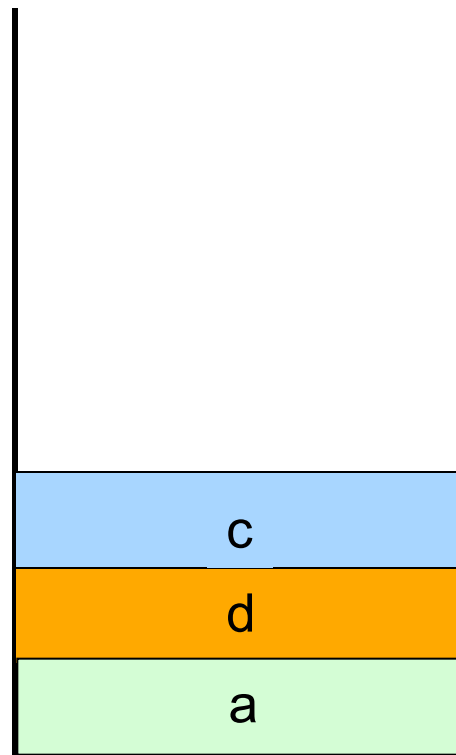
push d

push e

pop =>

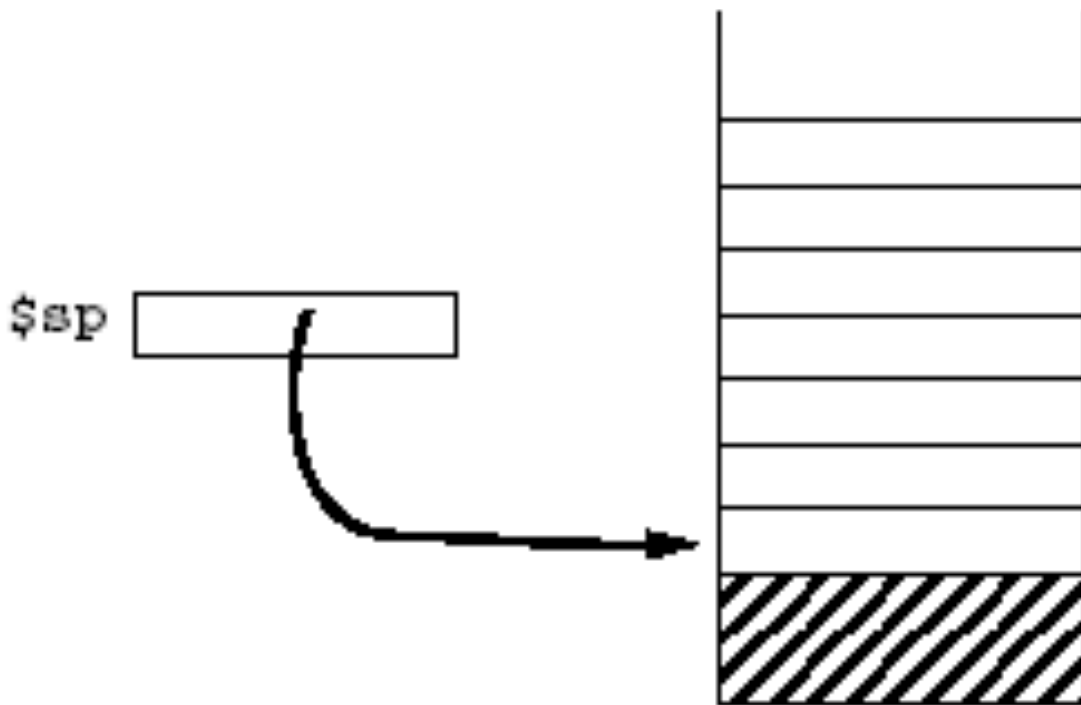
pop =>

pop =>



# MIPS Stack

- the MIPS system stack is in memory (the same memory as your program and data)
- register `$sp` contains the stack pointer
- the stack grows in the direction of smaller addresses
- the stack pointer always contains the address of the next free location



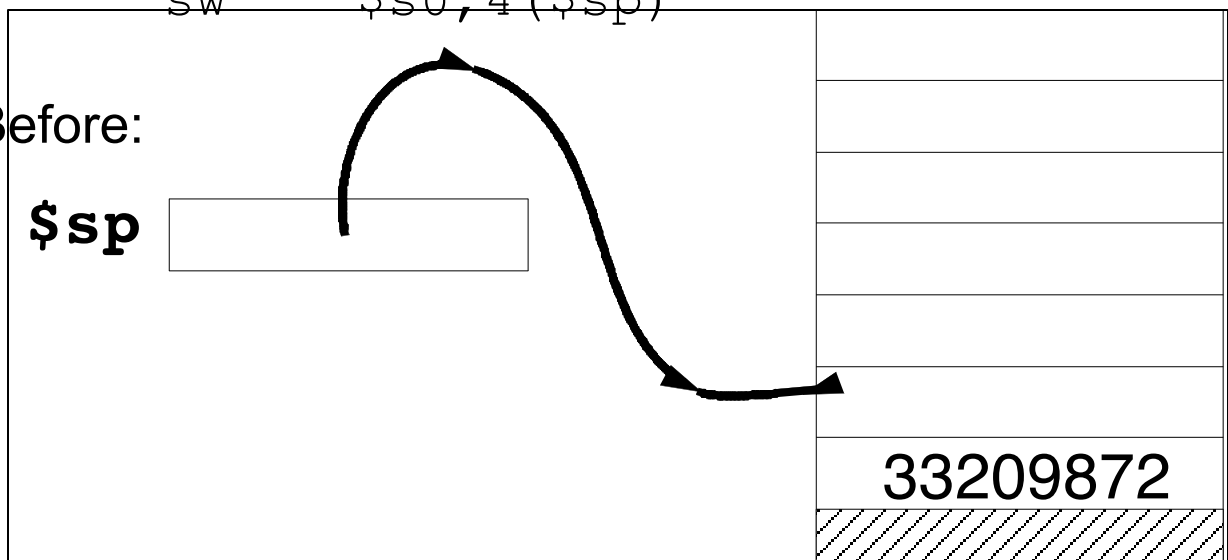
# Pushing

To push a word onto the stack:

```
sub    $sp, $sp, 4  
li     $s0, 0x12345678  
sw     $s0, 4($sp)
```

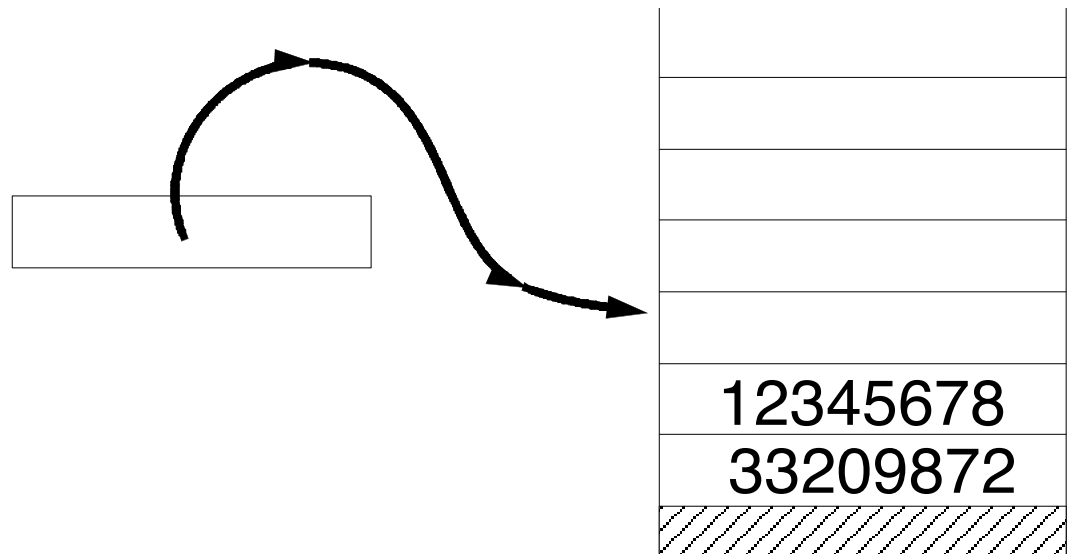
Before:

**\$sp**



After:

**\$sp**



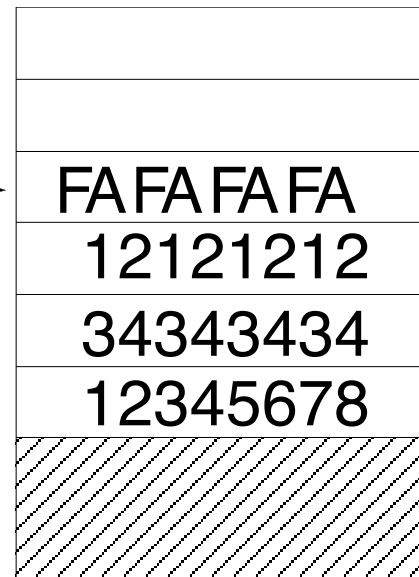
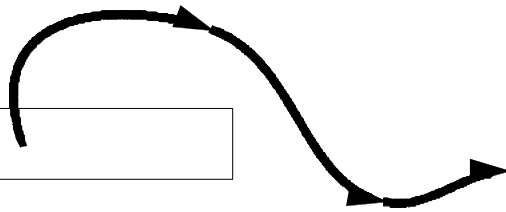
# Popping

To pop a word off the stack:

```
lw      $s0, 4($sp)
add     $sp, $sp, 4
```

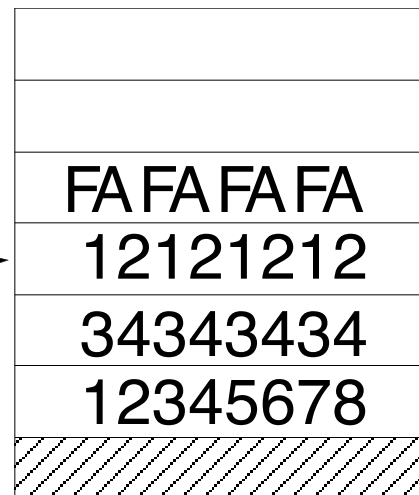
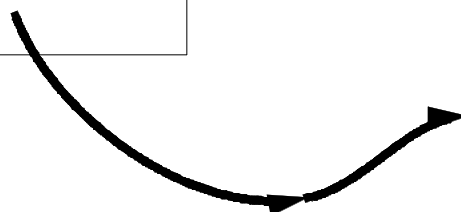
Before:

**\$sp**



After:

**\$sp**



# Saving Return Addresses

At the beginning of every procedure, push the return address on the stack. Then pop it at the end.

```
start:
    jal    mumbo
    ...
done
mumbo:
    sub    $sp,$sp,4    # push ra
    sw     $ra,4($sp)

    ...
    jal    jumbo
    ...
    lw     $ra,4($sp)   # pop ra
    add    $sp,$sp,4
    jr     $ra
```

Always do this!!

# Example Procedure

The 5 Levels Of <b>INCEPTION</b>				
LEVEL	WHO DREAMED IT?	WHO GOES THERE?	WHY ARE THEY THERE?	THE KICK
<b>LEVEL 1</b> <b>REALITY</b>	<b>No one...</b> We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
<b>LEVEL 2</b> <b>VAN CHASE</b>	<b>Yusuf</b> "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fischer Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
<b>LEVEL 3</b> <b>THE HOTEL</b>	<b>Arthur</b> "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.
<b>LEVEL 4</b> <b>SNOW FORTRESS</b>	<b>Eames</b> "The Forger"	Cobb, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. must be taken to the fort, where the idea they wish to plant will finally take hold.	Eames blows up the supports of the fortress, dropping it and causing freefall.
<b>LEVEL 5</b> <b>LIMBO</b>	<b>No one</b> It's a shared state	Cobb, Ariadne, Saito, Robert Fischer Jr. and Mal's projection	To get Fischer Jr. and Saito out.	Ariadne and Fischer fall off a building. Cobb and Saito shoot themselves.



# What is her name?

The 5 Levels Of <b>INCEPTION</b>				
LEVEL	WHO DREAMED IT?	WHO GOES THERE?	WHY ARE THEY THERE?	THE KICK
LEVEL 1 <b>REALITY</b>	No one... We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
LEVEL 2 <b>VAN CHASE</b>	Yusuf "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fischer Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
LEVEL 3 <b>THE HOTEL</b>	Arthur "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.
LEVEL 4 <b>SNOW FORTRESS</b>	Eames "The Forger"	Cobb, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. must be taken to the fort, where the idea they wish to plant will finally take hold.	Eames blows up the supports of the fortress, dropping it and causing freefall.
LEVEL 5 <b>LIMBO</b>	No one It's a shared state	Cobb, Ariadne, Saito, Robert Fischer Jr. and Mal's projection	To get Fischer Jr. and Saito out.	Ariadne and Fischer fall off a building. Cobb and Saito shoot themselves.

# S and T Registers

```
__start:
    lw    $t0, important_value
    jal   mumbo
    add   $t0, $t0, 1
    ...
done
```

# S and T Registers

There are two sets of general-purpose registers:

Saved registers:            \$s0-\$s8

Temporary registers:      \$t0-\$t9

Saved registers must be *preserved* across procedure calls, so if you use one in a procedure, you must restore its old value when you're done.

Temporary registers may *not* be *preserved* across procedure calls.

What's wrong with this picture?

```
__start:
    lw      $t0, important_value
    jal     mumbo
    add     $t0, $t0, 1
    ...
done
```

# Saving Registers

At the beginning of a procedure (after saving the return address) save any s registers you are going to use. Restore them at the end.

# s1 will hold the GNP  
# s2 will hold the avg. grease ratio  
# t0 is used for calculation

jumbo:

```
sub    $sp, $sp, 4      # push ra
sw     $ra, 4($sp)
sub    $sp, $sp, 4      # save $s1
sw     $s1, 4($sp)
sub    $sp, $sp, 4      # save $s2
sw     $s2, 4($sp)
...
lw     $s2, 4($sp)      # restore $s2
add    $sp, $sp, 4
lw     $s1, 4($sp)      # restore $s1
add    $sp, $sp, 4
lw     $ra, 4($sp)      # pop ra
add    $sp, $sp, 4
jr     $ra
```

Nobody said assembly language wasn't tedious.

# Saving Registers More Efficiently

We can make the previous example more efficient:

```
# s1: GNP
# s2: avg. grease ratio
# t0: used for calculation
```

jumbo:

```
sub    $sp,$sp,12
sw     $ra,12($sp) # push ra
sw     $s1,8($sp)  # save s1
sw     $s2,4($sp)  # save s2
...
lw     $s2,4($sp)  # restore $s2
lw     $s1,8($sp)  # restore $s1
lw     $ra,12($sp) # pop ra
add    $sp,$sp,12
jr     $ra
```

# How to move data?

The 5 Levels Of <b>INCEPTION</b>				
LEVEL	WHO DREAMED IT?	WHO GOES THERE?	WHY ARE THEY THERE?	THE KICK
<b>LEVEL 1 REALITY</b>	<b>No one...</b> We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
<b>LEVEL 2 VAN CHASE</b>	<b>Yusuf</b> "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fischer Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
<b>LEVEL 3 THE HOTEL</b>	<b>Arthur</b> "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.
<b>LEVEL 4 SNOW FORTRESS</b>	<b>Eames</b> "The Forger"	Cobb, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. must be taken to the fort, where the idea they wish to plant will finally take hold.	Eames blows up the supports of the fortress, dropping it and causing freefall.
<b>LEVEL 5 LIMBO</b>	<b>No one</b> It's a shared state	Cobb, Ariadne, Saito, Robert Fischer Jr. and Mal's projection	To get Fischer Jr. and Saito out.	Ariadne and Fischer fall off a building. Cobb and Saito shoot themselves.

# Passing Arguments: The Easy Way

Registers \$a0-\$a3 are reserved for passing arguments. They are *not preserved* across procedure calls.

Registers \$v0-\$v1 are for returning results.

```
# a0: one of the nums to be averaged
# a1: another num to be averaged
# v0: return the result
# t0: calculation
average:
    add    $t0,$a0,$a1
    div    $v0,$t0,2
    jr     $ra
```

What if we need to call another procedure?



# Passing Arguments: The General Way

In nested procedures, we may have to save argument values on the stack.

Sometimes, we'll have too many arguments to fit into 4 registers, or too many return values.

General Answer: use the stack.

- Caller pushes arguments and space for results.
- Callee uses arguments and fills in results.
- Caller pops everything.



# Arguments on the Stack

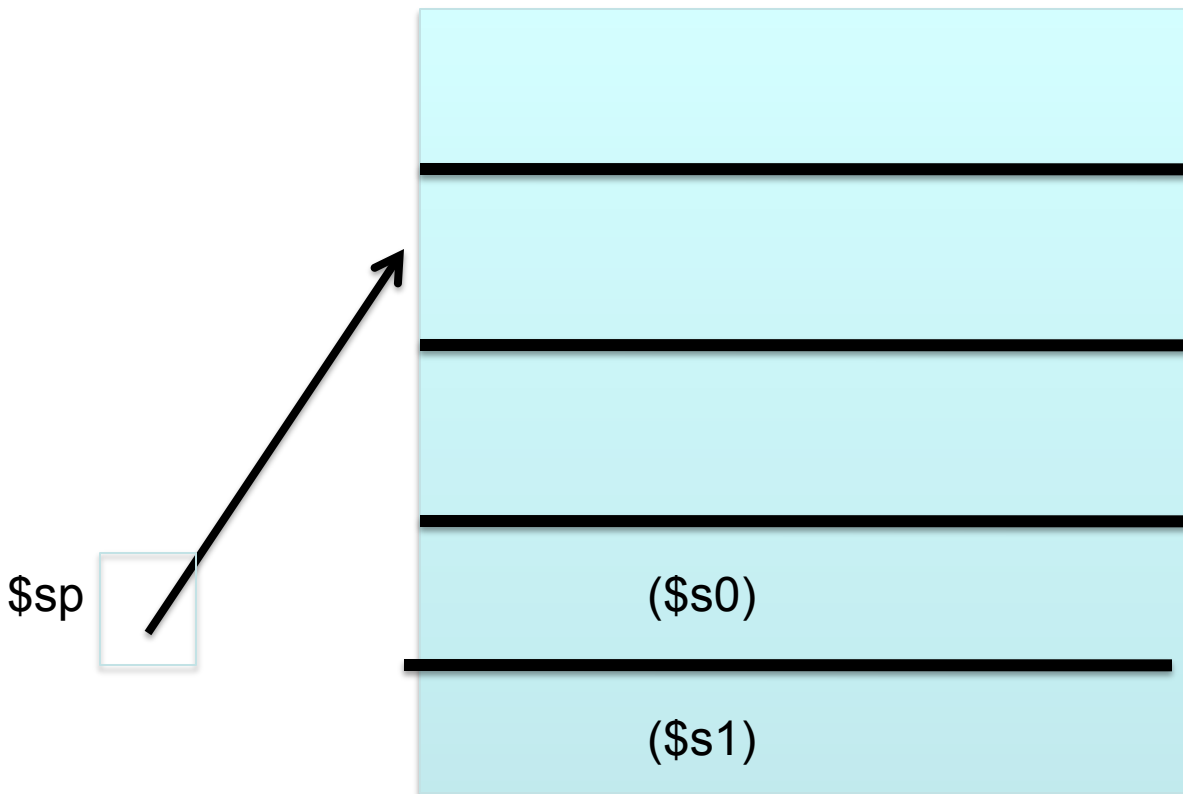
```
# average the values in $s0 and $s1, put the
# result in $s2

    sub    $sp,$sp,12      # space for rslt.
    sw     $s0,8($sp)      # push 1st param
    sw     $s1,12($sp)     # push 2nd param
    jal    average
    lw     $s2,4($sp)      # get result
    add    $sp,$sp,12
done

average:
    sub    $sp,$sp,4
    sw     $ra,4($sp)
    lw     $t0,12($sp)     # load 1st param
    lw     $t1,16($sp)     # load 2nd param
    add    $t0,$t0,$t1
    div    $t0,$t0,2
    sw     $t0,8($sp)      # store result
    lw     $ra,4($sp)      # pop ra
    add    $sp,$sp,4
    jr     $ra             # return
```

# Arguments on the Stack

```
# average the values in $s0 and $s1, put the
# result in $s2
    sub    $sp,$sp,12      # space for rslt.
    sw     $s0,8($sp)      # push 1st param
    sw     $s1,12($sp)     # push 2nd param
    jal    average
    lw     $s2,4($sp)      # get result
    add    $sp,$sp,12
done
```



# Arguments on the Stack

```
# average the values in $s0 and $s1, put the
# result in $s2

    sub    $sp,$sp,12      # space for rslt.
    sw     $s0,8($sp)      # push 1st param
    sw     $s1,12($sp)     # push 2nd param
    jal    average
    lw     $s2,4($sp)      # get result
    add    $sp,$sp,12
done

average:
    sub    $sp,$sp,4
    sw     $ra,4($sp)
```



# Arguments on the Stack

```
# average the values in $s0 and $s1, put the
# result in $s2
    sub    $sp,$sp,12      # space for rslt.
    sw     $s0,8($sp)      # push 1st param
    sw     $s1,12($sp)     # push 2nd param
    jal    average
    lw     $s2,4($sp)      # get result
    add    $sp,$sp,12
done

average:
    sub    $sp,$sp,4
    sw     $ra,4($sp)

    lw     $t0,12($sp)     # load 1st param
    lw     $t1,16($sp)     # load 2nd param

    add    $t0,$t0,$t1
    div    $t0,$t0,2
    sw     $t0,8($sp)      # store result
    lw     $ra,4($sp)      # pop ra
    add    $sp,$sp,4
    jr     $ra             # return
```

# Stack Allocation

Where are these variables allocated?

```
int fact(int n)
{
    if (n == 0)
        return 1;
    else {
        int f = fact(n-1);
        return n * f;
    }
}
```

# Stack Allocation

## Can we allocate an array on the stack?

```
# size of the array in $s1
# address of the array will be in $s2
```

```
mult $t0, $s1, 4
sub  $sp, $sp, $t0
add  $s2, $sp, 4
```

## How do I access the element i?

```
mult    $t1, "i", 4
add     $t1, $s2, $t1
lw      $t0, ($s2)
```

## You can do that in C/C++?

```
int* a = (int*) alloca(sizeof(int)*size);
```

## What is the life time of this array?

# Are we done?

“done” does not exist?

✓ just a short end for

```
jr $ra
```



But where are we jumping?